Продолжаем изучение переопределение операций для объектов классов языка C++. На этом занятии увидим, как и для чего выполняется переопределение встроенной операции присваивания.

Операция присваивания по умолчанию уже работает на уровне объектов любого класса и выполняет побайтное копирование содержимого одного объекта в другой. Но это не всегда желаемое поведение. Как пример, рассмотрим реализацию известной структуры данных – **динамического массива**.

Я напомню, что в основе динамического массива лежит обычный массив языка Си. Его состояние можно определить двумя целочисленными переменными:

- capacity – физическая длина массива;
- length – количество записанных в массив данных.

Как только длина length достигает размера capacity, то в памяти создается новый массив, часто удвоенной длины, в который копируются данные из прежнего массива. В итоге происходит автоматическое увеличение физической длины массива и новые данные заносятся уже в него. Вот в двух словах идея работы динамического массива.

Мы объявим класс DArray для работы с такой структурой данных в отдельном заголовочном **darray**.***h** следующим образом:

```language-cpp
class DArray {
    enum {
        start_length_array = 8, // начальная длина массива
        resize_factor = 2,      // множитель для увеличения длины массива
        max_length_array = 30,  // максимальная длина массива
        value_error = 2123456789, // специальное значение для обозначения ошибки данных
    };

    int* data {nullptr};
    int length {0};   // число записанных в массив значений
    int capacity {0}; // физический размер массива

    void _resize_array(int size_new); // увеличение размера массива data но не более max_length_array элементов
public:
```

```
    DArray() : length(0), capacity(start_length_array)
    {
        data = new int[start_length_array];
        capacity = start_length_array;
    }

    DArray(const DArray& other) : length(other.length), capacity(other.capacity)
    {
        data = new int[capacity];
        for(int i = 0; i < length;++i)
            data[i] = other.data[i];
    }

    ~DArray() { delete[] data; }

    int size() const { return length; }
    int capacity_ar() const { return capacity; }
    const int* get_data() const { return data; }

    const DArray& operator=(const DArray& other);

    void push_back(int value);
    int pop_back();
};
```

Так как каждый объект класса DArray формирует свой массив data, то необходимо переопределить:

- конструктор копирования;
- деструктор;
- операцию присваивания.

Вообще, в языке C++ 11-го стандарта существует так называемое **правило трех**. Оно гласит, если в классе переопределяется хотя бы один из приведенных трех методов, то следует явно переопределить и оставшиеся два. Как правило, они все работают в связке. И наш класс DArray – не исключение. Здесь действительно необходимо переопределить все эти три элемента. Для полноты картины отмечу, что начиная со стандарта C++14 правило трех превратилось в **правило пяти**, добавилось еще два метода:

- конструктор перемещения;
- конструктор копирования перемещением.

Давайте пропишем реализацию операции присваивания (и другие реализации методов) в файле **darray**.***cpp** следующим образом:

language-cpp

```cpp
#include "darray.h"

void DArray::_resize_array(int size_new)
{
    if(size_new <= capacity)
        return;

    while(capacity < size_new) {
        capacity *= resize_factor;
        if(capacity >= max_length_array) {
            capacity = max_length_array;
            break;
        }
    }

    int* p = new int[capacity];

    for(int i = 0;i < length; ++i)
        p[i] = data[i];

    delete[] data;
    data = p;
}

const DArray& DArray::operator=(const DArray& other)
{
    if(this == &other) // присваивание объекта самому себе
        return other;

    length = other.length;
    capacity = other.capacity;

    delete[] data;
    data = new int[capacity];
    for(int i = 0;i < length; ++i)
        data[i] = other.data[i];

    return *this;
}

void DArray::push_back(int value)
{
    if(length >= capacity) {
```

```
        _resize_array(capacity * resize_factor);
    }

    if(length < capacity)
        data[length++] = value;
}

int DArray::pop_back()
{
    if(length > 0)
        return data[--length];
    return value_error;
}
```

Как видите, операция присваивания работает вполне очевидным образом. Мы копируем все поля из присваиваемого объекта и дополнительно создаем свой массив data с копированием в него всех значений так же из присваиваемого объекта. Обратите внимание, на проверку в самом начале. Формально операцию присваивания можно записать с тем же самым объектом. А присваивать что-либо самому себе нет никакого смысла, да и программа работала бы некорректно. Поэтому было добавлено это условие. И, последнее замечание, операцию присваивания нельзя переопределять вне класса отдельной функцией, как это мы делали для операции сложения, только на уровне метода.

В результате, полученный класс можно использовать следующим образом:

language-cpp
```
#include <iostream>
#include "darray.h"

int main()
{
    DArray ar1, ar2;

    for(int i = 0;i < 10; ++i)
        ar1.push_back(i+1);

    ar2 = ar1;

    std::cout << ar2.size() << " " << ar2.capacity_ar() << std::endl;

    for(int i = 0;i < ar2.size(); ++i)
        std::cout << ar2.get_data()[i] << " ";
```

```
    return 0;
}
```

Однако у нас здесь не хватает очевидной операции для работы с объектами динамического массива – доступ к элементам по индексам. Например:

```language-cpp
 ar1[5] = 7;
    int v = ar1[3];
```

Но этот функционал мы добавим на следующем занятии.